REMARKS

The applicant has carefully considered the Office action dated July 25, 2007, and the references it cites. In the official action, all pending claims were rejected as being unpatentable over one or more of Hall (CPPROFJ: Aspect-capable Call Path Profiling of Multi-threaded Java Applications, 2002, IEEE), Hollingsworth (Critical Path Profiling of Message Passing and Shared Memory Program, Jan. 1998, IEEE), Broberg et al. (Performance Optimization Using Critical Path Analysis in Multithreaded Programs on Multiprocessors, 1999), and various technical articles in Intel Technology Journal (Feb. 2002) under 35 U.S.C. § 103(a). In view of the following remarks, reconsideration of the application is respectfully requested.

Claim 1 recites a method comprising, *inter alia*, monitoring information exchanged between a processing unit and first and second threads executed by the processing unit, determining a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree, and determining a wait time based on the information exchanged between the first and second threads. As explained below in detail, the applicants respectfully submit that the combination Hall and Hollingsworth does not teach or suggest monitoring information exchanged between a processing unit and first and second threads executed by the processing unit and determining a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

Hall describes aspect-capable call path profiling of multi-threaded java applications. The Office action contends that the abstract of Hall describes monitoring information exchanged between a processing unit and first and second threads executed by the processing unit. *See the official action dated July 25, 2007, page 3, § 3.* However, contrary to the assertion in the Office action, the abstract of Hall includes no such description. Rather, the

abstract of Hall describes call path profiling that "provides two different call path oriented views on program performance, a server view and a thread view." *See Hall, Abstract at ll. 10-12*. In fact, the abstract of Hall describes that the call path oriented views are to "incorporate a typed time notation for representing different program activities, such as wait and thread preemption times." *See Id. at ll. 14-16*. Hall further clarifies that the typed time notation determines the times that the threads are either running or waiting by "just randomly [stopping] the running program" and "[recording] the method running in each stack frame for the runtime stack of each thread[.]" *See Id. at page 6, § 5.1, ¶ 2*. A call stack is readily recognized as a collection of information about the active procedures and is generally used to return the execution of the program to a previous procedure that called a subsequent procedure. In other words, the call stack has no has no relation between the processor and the thread. Thus, it is respectfully submitted that the Hall abstract fails to meet the recitation of monitoring information exchanged between a processing unit and first and second threads.

Further, it is respectfully submitted that Hollingsworth does not cure these deficiencies. Therefore, for at the least these reasons, no combination of Hall and Hollingsworth describes or suggest monitoring information exchanged between a processing unit and first and second threads.

In rejecting claim 1, the Office action concedes that Hall does not describe determining a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree. To cure this deficiency, the Office action contends that Hollingsworth describes maintaining a critical path of thread execution. The Office action contends that Fig. 1 of Hollingsworth describes a critical path of thread execution. However, Hollingsworth describes Fig. 1 as "the sequence of events in a three process program" of a distributed computing program that passes messages between different processes. See

Hollingsworth *at page 1, § 2, ¶ 1*. Thus, Hollingsworth does not describe determining a critical path of thread execution and maintaining the critical path of thread execution.

Further, the Office action concedes that Hall does not cure this deficiency. Therefore, for at the least these reasons, no combination of Hall and Hollingsworth describes or suggests determining a critical path of thread execution and maintaining the critical path of thread execution in a critical path tree.

In addition, it is respectfully submitted that the combination of Hollingsworth with Hall would not be obvious because Hollingsworth teaches away from the alleged combination of Hall and Hollingsworth.

Hollingsworth is directed to critical path profiling of message passing in a distributed computing shared-memory program. In particular, the system of Hollingsworth is directed to an online (i.e., during program execution) profiling algorithm. Because the Hollingsworth system is performed during execution, Hollingsworth explicitly states that because "the number of nodes in the [program activity graph] is equal to the number of events during the program's execution, explicitly building the graph is not practical for long running program" and "[o]ne way to overcome this limitation is to develop an algorithm that does not require storing event logs or building the graph." *See Id. at page 1030, § 2.1, ¶ 3*. Thus, to reduce the amount of data, Hollingsworth actually opts to compute a portion of the critical path for a specific procedure. *See Id. at page 1030, § 2.1, ¶ 3*.

However, as described above, the Hall system is implemented by sampling the call stack, which is done by "randomly [stopping] the running program" and "[recording] the method running in each stack frame for the runtime stack of each thread" and then resuming execution of the program. *See Hall at page 6, § 5.1, ¶ 2*. Further, Hall describes repeatedly stopping the execution of the program and record large quantities of data "build up a

statistical picture of the program's behavior over the course of the run[.]" *See Id.* In other words, the Hollingsworth system explicitly teaches away from recording large quantities of data while the program is running and, in complete contrast, Hall explicitly describes recording the large quantities of data during the execution of the program. Thus, Hollingsworth explicitly teaches away from the combination of Hollingsworth and Hall.

For at least the foregoing reasons, it is respectfully submitted that it would not be obvious to combine the Hollingsworth with Hall. Thus, it is respectfully submitted that independent claim 1 and all claims dependent thereon are in condition for allowance.

Independent claims 11, 21, and 32 are also allowable for at least the same reasons discussed above in connection with claim 1. Therefore, it is respectfully submitted that independent claims 11, 21 and 32 and all claims dependent thereon are in condition for allowance.

The applicants respectfully submit that all claims are in condition for allowance. Reconsideration of the application and allowance thereof are respectfully requested. If there is any matter that the examiner would like to discuss, the examiner is invited to contact the undersigned representative at the telephone number set forth below.

Respectfully submitted,

HANLEY, FLIGHT & ZIMMERMAN, LLC
150 S. Wacker Dr., Suite 2100
Chicago, Illinois 60606

Dated: **January 25, 2007**

/Simon G. Booth/

Simon G. Booth
Reg. No. 58,582
Attorney for Applicants
312.580.1020